Improved Sequential MAP estimation of CABAC encoded data with objective adjustment of the complexity/efficiency tradeoff

S. Ben–Jamaa, Michel Kieffer, Pierre Duhamel

Efficient MAP estimation of CABAC encoded data with objective adjustment of the complexity/efficiency tradeoff

Salma Ben Jamaa, Michel Kieffer and Pierre Duhamel
LSS – CNRS – Supélec – Univ Paris-Sud,
3 rue Joliot-Curie - 91192 Gif-sur-Yvette cedex, France

Abstract

This paper presents an efficient MAP estimator for the joint source-channel decoding of data encoded with a context adaptive binary arithmetic coder (CABAC). The decoding process is compatible with realistic implementations of CABAC in standards like H.264, *i.e.*, handling adaptive probabilities, context modeling and integer arithmetic coding. Soft decoding is obtained using an improved sequential decoding technique, which allows to obtain various tradeoffs between complexity and efficiency. The algorithms are simulated in a context reminiscent of H264. Error detection is realized by exploiting on one side the properties of the binarization scheme and on the other side the redundancy left in the code string. As a result, the CABAC compression efficiency is preserved and no additional redundancy is introduced in the bit stream. Simulation results outline the efficiency of the proposed techniques for encoded data sent over AWGN and UMTS-OFDM channels.

Keywords: Arithmetic codes, Communication system performance, Error correction coding, Source coding

I. Introduction

An increasing number of applications require data transmission through noisy packet-switched channels, *e.g.*, wireless Internet. To overcome problems due to noise and packet

This work has been partly supported by the NEWCOM NoE. The authors would also thank France Telecom R&D, and especially I. Siaud and Dr M. Jeanne for providing the soft error patterns in the software platform of the NEWCOM NoE. Parts of this work haven been presented at [1] and [2]

losses, a popular approach is to use joint source-channel (JSC) decoding schemes, which maintain the efficiency in terms of data compression (since they decode the same bitstream as standard decoders) while improving robustness against errors. A significant part of the work in JSC decoding is related to the reliable decoding of variable length codes (VLC), see, *e.g.*, [3]–[6].

Arithmetic Coding (AC) [7] is currently the object of a growing interest as it yields higher compression efficiency when compared to other compression methods. Nevertheless, the high compression rate makes AC particularly vulnerable to transmission errors. This point has motivated the recent development of JSC decoding techniques for AC encoded data [8]–[13]. In most existing works, error detection and correction is usually performed by introducing redundancy in the compressed bitstream, thus reducing the compression efficiency. In [8], a *forbidden symbol* (FS) is introduced in the coding alphabet and used as an error detection device. This technique is coupled with ARQ in [10]. In [11], both depth first and breadth first decoding algorithms have been considered. Error detection is again achieved by testing the presence of a FS during the decoding process and error correction is based on a maximum-likelihood (ML) criterion. Sequential decoding with maximum *a posteriori* MAP estimation of the encoded sequence is combined with the use of a FS in [13]. In [14], a Trellis Coded Modulation is used jointly with AC; the FS is exploited to discard erroneous paths during a List Viterbi decoding process. The *quasi-arithmetic* coder used in [12] is represented by a finite state machine, where transitions between states are modeled by a Markov process. Redundancy is introduced by considering a reduced precision AC and adding synchronization markers before the encoding process. Recently, in [15], all sequences an integer AC may generate are represented by a 3D trellis taking into account the presence of a FS in the source alphabet. This trellis representation allows the use of soft output decoders such as the BCJR algorithm [16].

Context-based Adaptive Binary Arithmetic Coding (CABAC) [17] is the encoder used in the H.264/AVC standard [18]. By combining an adaptive binary encoding technique with context modeling, an improved compression efficiency is achieved especially in presence of non-stationary sources. For instance, when CABAC is used, the H.264 compression results are up to 20% better than those obtained by the baseline entropy coder of H.264/AVC, based on variable length codes [19]. However, since CABAC uses binary AC and probabilities stored in look-up tables, techniques similar to those cited above cannot be applied unless important changes are performed in the CABAC encoder implementation.

This work deals with the MAP estimation of CABAC encoded data. Its purpose is to provide an efficient MAP metric which is suited to an actual CABAC implementation, *i.e.*, a metric taking into account the delays between the code reception and the decoded stream emission due to the buffer-based integer implementation. For that purpose, an estimator (*observer*) of the CABAC encoder state is required at decoder side. No extra redundancy is compulsory: only redundancy introduced during the binarization step of the CABAC and left in the code string is exploited to detect errors. Our decoder has the characteristic of using an exact metric, in contrast with that proposed in [13], which would only be approximate on a realistic CABAC. Moreover, this exact computation is compatible with existing ways of adding redundancy, and a combination of both strategies would result in further improvements, but are not considered in this paper. Decoding is achieved using sequential decoders [20], where the improvement in terms of error resilience usually goes with an increase in decoding complexity. This problem is addressed here by the use of objective tests based on the MAP metric evaluation which allow an explicit adjustment of the tradeoff between decoding complexity and error correction efficiency.

Section II introduces the context of the work and recalls the main features of the CABAC implementation. Then, the way the exact MAP estimator is derived is explained in Section III. The decoding process is put at work by means of sequential decoding algorithms introduced in Section IV, where improvements provided to conventional sequential decoding schemes by allowing to adjust the complexity-efficiency tradeoff are presented. Finally, simulation results are shown in Section V.

II. Context of the work

This section first describes the considered transmission scheme and related notations. Then, the principles of arithmetic coding, derived from Elias Coding are recalled in Section II-B. The practical implementation of this basic version of AC is explained in Section II-C.

A. Transmission scheme

Data are assumed to be processed by a CABAC encoder, then packetized and transmitted over a radio mobile channel. Packets undergo some alteration during the transmission. This paper is concerned with the means of detecting and correcting transmission errors at the source bitstream level.

Since CABAC handles only binary data, a binarization step consisting in converting non-binary information into binary source symbols according to a *binarization scheme* [17] is needed. The encoder is then run, and sends packets of the resulting bitstream in the channel. Soft estimates are obtained from the output of the channel and fed to the CABAC decoder.

The input of the CABAC encoder is a sequence of K bins denoted by $S_1^K = \{S_1, ..., S_K\}$. It consists of a succession of binarized source symbols belonging to a set of binary words denoted by $\mathcal{C}$. *Bins* stand for bits obtained by the binarization process. The last binarized source symbol of S_1^K, denoted by EOS (End Of Sequence), indicates the end of the binarized stream. Let $X_1^N = \{X_1, ..., X_N\}$ be the succession of CABAC output bits. They are mapped using BPSK signaling into $R_1^N = \{R_1, ..., R_N\}$, with $R_i = \pm\sqrt{E_b}$, $i = 1 \dots N$, and transmitted within a single packet over a noisy memoryless channel. Let $Y_1^N = \{Y_1, ..., Y_N\}$ be the corresponding channel output.

Only N is assumed to be known at the decoder side. Capital letters are for random variables, and small letters for their values. Integers n and k denote respectively the current length of the received code stream (decoder input), and the current number of decoded bins (decoder output).

B. Basic principles of binary arithmetic coding (BAC)

Basic principles of BAC are recalled here, for more details, see, *e.g.,* [21].

At each iteration of the encoding process, a subinterval of $[0, 1)$ is recursively constructed. The current interval $[low, high)$ is divided into two subintervals of lengths L_0 and L_1, respectively proportional to the probabilities P_0 and P_1 of the source bins 0 and 1. One of these two subintervals is selected, depending on the value of the current bin. Once the last bin of EOS is encoded, the algorithm computes the real value V, belonging to the obtained interval, which can be represented by the minimum number of bits. The binary representation of V forms the code string. Based on V, the decoder is able to reconstruct intermediate intervals and to recover the source bin stream.

For sources with very dissymetric bin probabilities, and for long source sequences, L_0 or L_1 may get too small to be accurately handled by a finite-precision processor. This problem is solved with finite precision implementations of BAC.

C. Practical implementations of BAC

To overcome the precision problem, most AC encoders are implemented using integers [22]. Since this process is of interest in the derivation of the MAP decoder, see Section III, the corresponding algorithm is recalled below.

The initial interval $[0, 1)$ of reals is replaced by the interval $[0, 2^p)$ of integers, where $p \geqslant 2$ is the chosen finite precision, *i.e.*, p bits are used to represent low and $high$. Partition and selection are carried out every time a source symbol is encoded. Renormalization by doubling the size of the source interval is performed as long as one of the following conditions holds

1) If $high \leqslant 2^{p-1}$, low and $high$ are doubled.
2) If $2^{p-1} \leqslant high$, low and $high$ are doubled after subtracting 2^{p-1}.
3) If $2^{p-2} \leqslant low$ and $high < 3 \times 2^{p-2}$, low and $high$ are doubled after subtracting 2^{p-2}.

If the current interval (before renormalization) overlaps the midpoint of $[0, 2^p)$, no bit is output. The number of consecutive times this occurs is stored in a variable called $follow$. If the current interval (before renormalization) lies entirely in the upper or lower half of $[0, 2^p)$, the encoder emits the leading bit of low (0 or 1) and $follow$ opposite bits (1 or 0). This is called the *follow-on* procedure [22].

At decoder side, a p-bit buffer $value$, formed by the p first received bits, is used to determine the divisions and selections of $[0, 2^p)$ which have been performed at encoder side. Decoding starts after an initialization step, consisting in loading p bits into $value$ without outputting any estimate of the source bins. The partition and selection operations performed at encoder side are determined using $value$, which helps thus estimating the source bins. Every renormalization results in multiplying by 2 the integer associated to $value$, maybe after subtracting 2^{p-1} or 2^{p-2}, and thus shifting the buffer $value$ to the left. Therefore, the next bit in the code stream is loaded to occupy the vacant position on the right of the buffer. This is repeated until all the code stream has passed into the buffer.

In the context-based BAC, interval divisions are performed according to bins probabilities deduced from *contexts*. Context modeling consists in assigning probability distribution models to the binary source symbols according to statistical dependencies between them. Each context is characterized by its *index*, *state* and the value of the most probable bin (MPS: Most Probable Symbol) [17]. The index may be specified by the current bin position in the binary source symbol and by the type of the *syntax element* to be encoded. The state indicates the probability estimate of the least probable bin (LPS: Least Probable Symbol), and is updated

iteratively. Selection depends on whether the current bin is the MPS or the LPS.

III. MAP ESTIMATION OF CABAC ENCODED DATA

In [13], a MAP estimator is proposed to evaluate the length K and content of the CABAC encoder input bin sequence s_1^K using y_1^N. This estimator may be expressed as

$$\begin{aligned}\left(\widehat{K}, \widehat{s}_1^{\widehat{K}}\right) &= \arg\max_{k,s_1^k} P\left(S_1^k = s_1^k | Y_1^N = y_1^N\right) \\ &= \arg\max_{k,s_1^k} \frac{P\left(S_1^k = s_1^k\right) P\left(Y_1^N = y_1^N | S_1^k = s_1^k\right)}{P\left(Y_1^N = y_1^N\right)}. \end{aligned} \quad (1)$$

The main difficulty in using (1) lies in the evaluation of $P\left(Y_1^N = y_1^N | S_1^k = s_1^k\right)$. The problem is circumvented in [13] by using a modified version of the previous estimator

$$\left(\widehat{K}, \widehat{s}_1^{\widehat{K}}, \widehat{x}_1^N\right) = \arg \max_{\substack{k,s_1^k,x_1^N \\ e\left(s_1^k\right)=x_1^N}} \frac{P\left(S_1^k = s_1^k\right) P\left(Y_1^N = y_1^N | X_1^N = x_1^N\right)}{P\left(Y_1^N = y_1^N\right)}, \quad (2)$$

where $e\left(s_1^k\right)$ represents the output of the CABAC encoder fed with the bin sequence s_1^k. The maximisation is thus performed among all sequences which are valid outputs of the CABAC encoder. Nevertheless, the set on which the maximisation is performed is quite complex, and may be described by a trellis of reasonable size only for reduced-complexity AC, such as those presented in [12]. For realistic implementations, sequential decoders (see [20] and Section IV) have to be put at work. These decoders perform an iterative decoding using only a part of the observations. For example, in [13], at step n of the decoding process, (2) is replaced by

$$\widehat{x}_1^n = \arg \max_{x_1^n \in B_n} \frac{P\left(S_1^{k\left(x_1^n\right)} = d\left(x_1^n\right)\right) P\left(Y_1^n = y_1^n | X_1^n = x_1^n\right)}{P\left(Y_1^n = y_1^n\right)}, \quad (3)$$

where $d\left(x_1^n\right)$ represent the output (formed by $k\left(x_1^n\right)$ bins) of the CABAC decoder fed with x_1^n at its input. In (3), B_n is the set containing the n bits long prefixes of all sequences that could be generated by the CABAC encoder described in Section II-C.

As will become clear at the end of Section III-B, the estimate provided by (3) is suboptimal when compared to the *a posteriori* estimate of x_1^n which could be obtained using y_1^n and the fact that x_1^n is the output of a CABAC encoder.

In [9], Sayir proposes a MAP estimation of noisy arithmetic encoded data, which has inspired that work, though the decoding delay occurring at the CABAC decoder is not taken into account in [9]. In our work, a different definition of the MAP estimate is adopted. As

will be shown, this requires the estimation of some features of the encoder state ($follow$ and probability estimates) at decoder side.

A. Definition and assumptions

The aim of this work is to determine at any $n \leqslant N$, the sequence $\widehat{x}_1^n$ maximizing the *a posteriori* probability (APP) given by

$$\widehat{x}_1^n = \arg \max_{x_1^n \in B_n} P(X_1^n = x_1^n | Y_1^n = y_1^n), \tag{4}$$

As in (3), the evaluation of (4) takes into account the fact that x_1^n is the output of a CABAC encoder fed with binarized source symbols. Once N is reached, $\widehat{x}_1^N$, maximizing (4) for $n = N$, may be decoded as $\widehat{s}_1^{\widehat{K}}$ satisfying all constraints imposed by the binarization. As there is a one-to-one mapping between $\widehat{x}_1^N$ and $\widehat{s}_1^{\widehat{K}}$, $\widehat{s}_1^{\widehat{K}}$ also maximizes (1).

Consider the decoder input sequence x_1^n, and the sequence of bins $s_1^{k\left(x_1^n\right)} = d\left(x_1^n\right)$ obtained at the output of a CABAC decoder fed with x_1^n (to make notations shorter, $k\left(x_1^n\right)$ will be denoted by k). The decoder input sequence x_1^n may then be decomposed into three parts:

1) $x_1^{n'(s_1^k)} = e\left(s_1^k\right)$ is the code string that would be emitted by an encoder fed with $s_1^k = d\left(x_1^n\right)$, $n'(s_1^k)$ being its length,
2) $x_{n'(s_1^k)+1}^{n'(s_1^k)+F(s_1^k)}$ are the *postponed bits* [9], *i.e.*, the $F(s_1^k)$ first bits that would be emitted during the next follow-on procedure by an encoder fed with s_1^k. The number $F(s_1^k)$ of postponed bits is equal to $follow + 1$, keeping in mind that $follow$ depends on s_1^k. Postponed bits may only take the values $\{1, 0, \ldots, 0\}$ and $\{0, 1, \ldots, 1\}$, depending on the coder internal state (low, $range$, $follow$, contexts) after being fed with s_1^k,
3) $x_{n'(s_1^k)+F(s_1^k)+1}^n$, are bits assumed independent of s_1^k.

As $n'(s_1^k)$, $F(s_1^k)$, and the current probability estimates are not directly available at the decoder (due to delays introduced by the encoding and decoding buffers), a possible way to estimate them is to *reencode* s_1^k. The additional encoder used for that purpose is called *observer* to avoid confusion with the encoder at emitter side, see Figure 1. Moreover, as X_1^N is the output of an arithmetic encoder, the X_i, $i = 1 \ldots N$, are assumed equally likely,

$$P(X_i = 1) = P(X_i = 0) = \frac{1}{2}, \forall i \in \{1 \ldots N\}. \tag{5}$$

In the remainder of the paper, the notations n', k, and F are adopted for $n'(s_1^k)$, $k\left(x_1^n\right)$, and $F(s_1^k)$ when there is no risk of confusion. Moreover, the random variables X, Y, and S are omitted in the expressions of the probabilities, provided that there is no ambiguity.

B. APP metric derivation

The APP in (4) may be written as

$$P(X_1^n = x_1^n | Y_1^n = y_1^n) = P(x_1^n) \frac{P(y_1^n|x_1^n)}{P(y_1^n)}. \tag{6}$$

To evaluate (6), one has to express the *a priori* probability $P(x_1^n)$ in terms of *a priori* probabilities of the source bins.

For $s_1^k = d(x_1^n)$, one has

$$P\left(S_1^k = d(x_1^n) | X_1^n = x_1^n\right) = 1. \tag{7}$$

As

$$\begin{aligned} P\left(S_1^k = d(x_1^n) | x_1^n\right) &= P\left(S_1^k = d(x_1^n) | x_1^{n'}, x_{n'}^n\right) \\ &= \frac{P\left(S_1^k = d(x_1^n) | x_{n'}^n\right) P\left(x_1^{n'} | x_{n'}^n, S_1^k = d(x_1^n)\right)}{P\left(x_1^{n'} | x_{n'}^n\right)}, \end{aligned}$$

using (7), one gets

$$P\left(x_1^{n'} | x_{n'}^n\right) = P\left(S_1^k = d(x_1^n) | x_{n'}^n\right) P\left(x_1^{n'} | x_{n'}^n, S_1^k = d(x_1^n)\right).$$

Since $e(d(x_1^n))$ yields $x_1^{n'}$ (see Section III-A), $P\left(x_1^{n'} | x_{n'}^n, S_1^k = d(x_1^n)\right) = 1$ and one gets

$$P\left(x_1^{n'} | x_{n'}^n\right) = P\left(S_1^k = d(x_1^n) | x_{n'}^n\right). \tag{8}$$

Therefore, using the fact that $P(x_1^n) = P(x_1^{n'}, x_{n'+1}^n)$, (8) leads to

$$\begin{aligned} P(x_1^n) &= P\left(S_1^k = d(x_1^n) | x_{n'}^n\right) P\left(x_{n'+1}^n\right) \\ &= P\left(S_1^k = d(x_1^n), X_{n'}^n = x_{n'}^n\right). \end{aligned} \tag{9}$$

Using (9), (6) may be rewritten as

$$\begin{aligned} P(x_1^n | y_1^n) &= P\left(S_1^k = d(x_1^n), X_{n'}^n = x_{n'}^n\right) \frac{P(y_1^n|x_1^n)}{P(y_1^n)} \\ &= P\left(S_1^k = d(x_1^n)\right) P\left(x_{n'+1}^n | S_1^k = d(x_1^n)\right) \frac{P(y_1^n|x_1^n)}{P(y_1^n)}. \end{aligned} \tag{10}$$

Now, the decomposition of x_1^N into three parts may be further exploited to obtain a computable APP. The second term of the right part of (10) may be written as

$$\begin{aligned} P\left(x_{n'+1}^n | S_1^k = d(x_1^n)\right) &= P\left(x_{n'+1}^{n'+F}, x_{n'+F+1}^n | S_1^k = d(x_1^n)\right) \\ &= P\left(x_{n'+F+1}^n | S_1^k = d(x_1^n)\right) P\left(x_{n'+1}^{n'+F} | x_{n'+F+1}^n, S_1^k = d(x_1^n)\right) \end{aligned} \tag{11}$$

As $x_{n'+F+1}^{n}$ does not depend on $s_1^k = d\left(x_1^n\right)$ (see Section III-A),

$$P\left(x_{n'+F+1}^{n} | S_1^k = d\left(x_1^n\right)\right) = P\left(x_{n'+F+1}^{n}\right). \tag{12}$$

Moreover, assuming that $X_{n'+F+1}^{n}$ does not provide more information about $X_{n'+1}^{n'+F}$ than S_1^k does, one gets

$$P\left(x_{n'+1}^{n'+F} | x_{n'+F+1}^{n}, S_1^k = d\left(x_1^n\right)\right) = P\left(x_{n'+1}^{n'+F} | S_1^k = d\left(x_1^n\right)\right). \tag{13}$$

Finally, combining (10), (11), (12), and (13), the APP (6) may be expressed as

$$P\left(x_1^n | y_1^n\right) = P\left(S_1^k = d\left(x_1^n\right)\right) P\left(x_{n'+F+1}^{n}\right) P\left(x_{n'+1}^{n'+F} | S_1^k = d\left(x_1^n\right)\right) \frac{P(y_1^n | x_1^n)}{P(y_1^n)}. \tag{14}$$

When compared to the metric adopted in [13], two additional terms are involved in (14), namely $P\left(x_{n'+F+1}^{n}\right)$ and $P\left(x_{n'+1}^{n'+F} | S_1^k = d\left(x_1^n\right)\right)$. The evaluation of the probabilities making up (14) is described in Section III-C .

C. Practical implementation of the MAP estimator

In this section, we show the way the MAP estimate (4) is computed and updated using (14) for a CABAC decoder.

1) Evaluation of the bin stream probability $P\left(S_1^k = d\left(x_1^n\right)\right)$: Here, a priori probabilities of the form $P(S_1^k = s_1^k)$ have to be evaluated. The source bin stream is a sequence consisting of a succession of binarized symbols belonging to $\mathcal{C}$. The probability of each source symbols may be estimated by algorithms such as the one described in [23], or assumed to be known and deduced from the encoding tree as in [24], [25].

As this paper deals with an adaptive context based AC, one can rely on the adaptive probabilities of the MPS and the LPS deduced from the contexts at the decoder side. Indeed,

$$P(S_1^k = s_1^k) = P(S_1 = s_1) \prod_{i=2}^{k} P\left(S_i = s_i | S_1^{i-1} = s_1^{i-1}\right),$$

where $P\left(S_i = s_i | S_1^{i-1} = s_1^{i-1}\right)$ is estimated by P_{LPS} if $s_i = \text{LPS}$ and $1 - P_{\text{LPS}}$ if $s_i = \text{MPS}$.

When incomplete binarization schemes are considered (*i.e.*, when the Kraft inequality is strict), for some s_1^k, one may get

$$P(S_k = s_k | S_1^{k-1} = s_1^{k-1}) = 0, \tag{15}$$

which corresponds to values of s_k not consistent with the binarization tree.

2) Evaluating probabilities related to the transmission channel: The likelihood $P(y_1^n|x_1^n)$ and the channel output probability $P(y_1^n)$ are deduced from the channel model. In Section V, the AWGN channel model and an UMTS-OFDM channel model are considered. Both channels being assumed memoryless, one may write

$$P(y_1^n|x_1^n) = \prod_{i=1}^{n} P(y_i|x_i). \tag{16}$$

For an AWGN channel, $P(y_1^n)$ can be approximated by 2^{-n}, as in [13], and the last term of (14) becomes

$$\frac{P(y_1^n|x_1^n)}{P(y_1^n)} = \frac{2^n}{(2\pi\sigma^2)^{n/2}} \prod_{i=1}^{n} \exp\left(\frac{-(r_i - y_i)^2}{2\sigma^2}\right),$$

where $r_i = \pm\sqrt{E_\mathrm{b}}$, depending of x_i. When for each bit at channel output a Log Likelihood Ratio (LLR) is available, for the i-th received bit one has

$$LLR_i = \log \frac{P(X_i = 1|y_i)}{P(X_i = 0|y_i)}.$$

Using (5), one gets

$$P(X_i = 1|y_i) = \frac{1}{\exp(-LLR_i) + 1} \tag{17}$$

$$P(X_i = 0|y_i) = \frac{1}{\exp(LLR_i) + 1}. \tag{18}$$

Consequently, for such channels, one obtains

$$\frac{P(y_1^n|x_1^n)}{P(y_1^n)} = \prod_{i=1}^{n} \frac{2}{\exp\left(-\frac{r_i}{\sqrt{E_b}} LLR_i\right) + 1}.$$

3) Evaluating $P\left(x_{n'+F+1}^n\right)$: All sequences $x_{n'+F+1}^n$ are assumed to have equal *a priori* probability, *i.e.*, $P(x_{n'+F+1}^n) = 2^{-n+(n'+F+1)}$. If more information is available (provided in the context of iterative decoding, for example), it may obviously be used.

4) Evaluating the postponed bits probability $P\left(x_{n'+1}^{n'+F}|S_1^k = d\left(x_1^n\right)\right)$: In this case, the $k\left(x_1^n\right)$ first encoder input bins are assumed to be equal to $s_1^k = d\left(x_1^n\right)$. Under that hypothesis, the aim is to determine the probability that $X_{n'+1}^{n'+F} = x_{n'+1}^{n'+F}$. F represents the number of bits to be output as soon as a follow-on procedure is performed. More bins $s_{k+1}^{k'}$ are necessary to perfectly determine $x_{n'+1}^{n'+F}$ at the observer output. A possible way to estimate $P\left(x_{n'+1}^{n'+F}|S_1^k = s_1^k\right) = P\left(x_{n'+1}^{n'+F}|s_1^k\right)$ is to write this probability as

$$P\left(x_{n'+1}^{n'+F}|s_1^k\right) = \sum_{s_{k+1}^{k'}} P\left(s_{k+1}^{k'}|s_1^k\right) P(x_{n'+1}^{n'+F}|s_{k+1}^{k'}, s_1^k),$$

which requires to feed the observer with all possible $s_{k+1}^{k'}$ until a follow-on procedure is performed.

This procedure has a combinatorial complexity. Here, the procedure for a single additional bin s is described, its generalization to more bins is straightforward.

For a given value of $k\left(x_1^n\right)$, the tail of s_1^k corresponds to the beginning of a binarized source symbol. According to the binarization scheme, the following bin may take two values (MPS or LPS). If incomplete binarization schemes are involved, only one value of S_{k+1} may be possible. Considering both cases $S_{k+1} = \text{MPS}$ and $S_{k+1} = \text{LPS}$, $P\left(x_{n'+1}^{n'+F}|s_1^k\right)$ becomes,

$$\begin{aligned} P\left(x_{n'+1}^{n'+F}|s_1^k\right) &= P_{\text{MPS}} P\left(x_{n'+1}^{n'+F}|S_{k+1} = \text{MPS}, s_1^k\right) \\ &+ P_{\text{LPS}} P\left(x_{n'+1}^{n'+F}|S_{k+1} = \text{LPS}, s_1^k\right), \end{aligned}$$

where $P_{\text{MPS}} = P(S_{k+1} = \text{MPS}|s_1^k)$ and $P_{\text{LPS}} = P(S_{k+1} = \text{LPS}|s_1^k)$ are deduced from the contexts estimated at the observer. If postponed bits are emitted, their values may be either $\{0, 1, \ldots, 1\}$ or $\{1, 0, \ldots, 0\}$. If $x_{n'+1}^{n'+F}$ is not equal to one of these two sequences, one has

$$P(x_{n'+1}^{n'+F}|s_1^k) = 0. \tag{19}$$

Now, if $x_{n'+1}^{n'+F}$ is equal to $\{0, 1, \ldots, 1\}$ or $\{1, 0, \ldots, 0\}$, and

- only a MPS produces $x_{n'+1}^{n'+F}$ then $P(x_{n'+1}^{n'+F}|s_1^k) = P_{\text{MPS}}$,
- only a LPS produces $x_{n'+1}^{n'+F}$ then $P(x_{n'+1}^{n'+F}|s_1^k) = P_{\text{LPS}}$,
- both MPS and LPS produce $x_{n'+1}^{n'+F}$ then $P(x_{n'+1}^{n'+F}|s_1^k) = 1$.

In all other cases, it is assumed that $P(x_{n'+1}^{n'+F}|s_1^k) = \frac{1}{2}$.

Practical implementations could fed the observer with more than a single bit to improve the evaluation of $P(x_{n'+1}^{n'+F}|s_1^k)$. In practical situations, satisfactory results are obtained with less than 3 additional bits. The additional complexity remains thus limited.

IV. Sequential decoding

In order to compute the MAP estimator (14), the APP has to be evaluated for all paths of the decoding tree. To an observation Y_1^n, one may assign up to 2^n paths representing possible estimates of the code string X_1^n. For relatively large n, examining the whole decoding tree is infeasible. The purpose of sequential decoders is to find the best path, according to a chosen metric, without examining too many branches. The most popular sequential decoding algorithms are the stack algorithm (SA) [26], [27], the M-algorithm (MA) [20] and their variants, such as the *generalized stack algorithm* [28].

A. Stack Algorithm

The stack algorithm (SA) [26], [27] is a *metric first* search performed iteratively: an ordered stack containing previously examined paths (of different lengths) is maintained. The best path (located in the top of the stack) is expanded by the exploration of the two branches following the current node. The top path is then removed and the two new paths are merged in the stack.

Algorithm 1: Basic Stack Algorithm	
Step 1.	Initialize the stack with a single line containing the root of the decoding tree.
Step 2.	Extend the top path by creating two new paths, one for each possible value of the following emitted bit.
Step 3.	Store the two created paths in the stack with their corresponding metric. Sort the stack according to the metric. Drop paths lying beyond the stack size limit.
Step 4.	If the top path reaches the maximum depth on the decoding tree, stop. Otherwise, go to 2.

Many variations of this basic version have been proposed in the literature, such as the *generalized stack algorithm* [28], which extends the first $\ell \geq 2$ paths in the stack instead of only the first one. This allows packet loss reduction especially when the beginning of the received sequence is strongly disturbed. The *multiple stack algorithm* [29] is another approach claiming theoretically packet loss free decoding.

B. M-Algorithm

The M-algorithm (MA) is a *breadth first* search, the breadth being M. The decoding tree exploration is performed by iteratively incrementing the depth of the paths, keeping only the best M paths according to the metric.

Algorithm 2: Basic M-Algorithm	
Step 1.	Initialize the list with a single line containing the root of the decoding tree to which a null metric is assigned.
Step 2.	Extend all paths to the following branches, creating 2 new paths from each stored path.
Step 3.	Keep only the M best paths in terms of metric.
Step 4.	Stop if one of the M maintained paths reaches the maximum depth. Otherwise, go to 2.

C. Adapting sequential decoding to CABAC

In this section, the proposed sequential decoder is detailed. First, a metric based on the *a posteriori* probability derived in Section III-B is adopted. Then, drop conditions are put at work in order to lighten the decoding tree and avoid false tracks. These conditions can be considered as tools to detect errors in some transmitted bitstreams, without any additional redundancy. Finally, the way the decoding process stops is explained.

1) Decoding metric: Let $I(it)$ be the set of paths maintained at the it-th iteration of a sequential decoding algorithm. For a breadth-first search, all stored paths in $I(it)$ have the same length $n = it$. For a metric first search, paths with unequal lengths may be considered. Thus, both length n and values of bits composing x_1^n vary from one path belonging to $I(it)$ to an other. The metric assigned to every path x_1^n is derived from (14) and given by

$$\mathcal{M}_{\text{MAP}}(x_1^n) = \log P(x_1^n | y_1^n). \tag{20}$$

The best estimator of the beginning of X_1^N at the it-th iteration is the path $\widehat{x}_1^n$ maximizing the APP among all paths belonging to $I(it)$, and given by

$$(\widehat{x}_1^n)_{it} = \arg \max_{x_1^n \in I(it)} \mathcal{M}(x_1^n).$$

2) Drop conditions: Drop conditions allow the decoder to identify some paths not deserving to be stored. Such paths are removed from the decoding tree. The most used drop conditions are those related to the complexity restriction as a maximum number of simultaneously stored paths is fixed. When this number is reached, any path having a metric smaller than the metric of the last path is dropped. If the correct path is dropped in this way, the decoding fails and decoder may not output any solution. This event is called *erasure*. Note, however, that this is a somewhat different definition compared to the classical one.

A constraint on s_1^K results from the assumption that the length of the code string (N bits) is known and that the last binarized symbol is the EOS symbol. This constraint leads to two dropping situations. First, if a path x_1^N of N bits is examined over the decoding tree, and if the associated s_1^k is not ended by EOS, the sequence s_1^k is then deemed erroneous and x_1^N is dropped. Second, if s_1^k contains EOS while less than N bits have been decoded, an error is detected and the path is dropped. In these cases, paths are dropped without any risk of packet loss as the decoded stream cannot correspond to $\widehat{s}_1^{\widehat{K}}$.

Other drop conditions are directly deduced from the metric evaluation. Indeed, when the APP probability of a given path is equal to zero, it is very likely to be automatically discarded when explored paths are sorted and that the constraint on the maximum number of stored paths is applied. The APP can be equal to zero in the following cases: First, as CABAC mostly relies on incomplete binarization schemes (single infinite extended binarized symbol set consisting of zero order Exp-Golomb codes [17]), the situation described in (15) is likely to occur leading to a null APP. Second, when the case described by (19) occurs, a null APP is assigned to the corresponding path on the decoding tree. Notice that this drop condition exploits the little amount of redundancy left by the CABAC encoder in the semantic of the code string.

3) Stop conditions: The decoding tree exploration stops when a path x_1^N of N bits yields a sequence of binarized symbols s_1^k ending with EOS. Then, x_1^N is the path maximizing $\mathcal{M}_{MAP}(x_1^N)$ among all the stored paths at the current iteration. The sequence s_1^k is the solution output by the decoder. In some cases, decoding stops when all paths have been dropped, or when the maximum allowed computational effort has been reached. Both of these situations may lead to the occurrence of an erasure. The lost packet may be re-emitted if ARQ is allowed.

D. Objective adjustment of the efficiency-complexity tradeoff

The decoding performance in terms of error resilience efficiency is improved when more paths are explored in the decoding tree. This is especially true when the beginning of the code string is strongly corrupted. Nevertheless, this improvement usually goes with increasing decoding complexity. Therefore, in order to judiciously adjust a tradeoff between decoding complexity and efficiency, an objective test providing a new drop condition is proposed.

Assume that the path to be extended is x_1^{n-1}. Two extensions can be considered, $x_n = 1$ and $x_n = 0$. The hard decoder discards systematically one extension and thus, explores only

one path on the decoding tree. The idea is to derive a test allowing to decide if both of the two choices deserve being considered or if a hard decision on the current bit is sufficient. In [11], a 2Δ width *null zone* is used to determine whether a hard decision on y_n is reliable, error detection is then performed by means of a FS. Here, we consider that if a path has a relatively low metric, it is very likely to be dropped in the next iterations. Thus, omitting exploring and storing such paths saves computational effort. The idea is to derive an objective criterion in order to characterize a *low* metric for a controlled amount of decision errors.

At the node corresponding to x_1^{n-1} on the decoding tree, three actions may be taken:

- A_0^n : only the branch corresponding to $x_n = 0$ is explored.
- A_1^n : only the branch corresponding to $x_n = 1$ is explored.
- $A_{0|1}^n$: both extensions are explored.

Let

$$\Lambda_n = \mathcal{M}_{\text{MAP}}(x_1^{n-1}, 1) - \mathcal{M}_{\text{MAP}}(x_1^{n-1}, 0)$$

be the logarithm of the *a posteriori* probability ratio. The purpose is to derive a threshold T such that

$$\Lambda_n \underset{A_1^n}{\overset{A_{0|1}^n}{\lessgtr}} T \tag{21}$$

$$-\Lambda_n \underset{A_{0|1}^n}{\overset{A_0^n}{\lessgtr}} T \tag{22}$$

According to (20), and using the assumption that the channel is memoryless, one may write

$$\begin{aligned} \Lambda_n &= \log P(x_1^{n-1}, X_n = 1|y_1^n) - \log P(x_1^{n-1}, X_n = 0|y_1^n) \\ &= \log P(y_n|X_n = 1) - \log P(y_n|X_n = 0). \end{aligned} \tag{23}$$

The probability of error P_e, corresponding to the probability of loosing the correct path, may be evaluated as

$$\begin{aligned} P_e &= P(A_0^n, X_n = 1) + P(A_1^n, X_n = 0) \\ &= P(\Lambda_n < -T, X_n = 1) + P(\Lambda_n > T, X_n = 0) \\ &= P(X_n = 1)P(\Lambda_n < -T|X_n = 1) + P(X_n = 0)P(\Lambda_n > T|X_n = 0) \\ &= \frac{1}{2}P(\Lambda_n < -T|X_n = 1) + \frac{1}{2}P(\Lambda_n > T|X_n = 0). \end{aligned}$$

Similarly, the probability P_f of useless extension of a the path with two branches is

$$\begin{aligned} P_f &= P(A_{0|1}, X_n = 1) + P(A_{0|1}, X_n = 0) \\ &= \frac{1}{2}P(-T < \Lambda_n < T|X_n = 1) + \frac{1}{2}P(-T < \Lambda_n < T|X_n = 0). \end{aligned}$$

To get T, one minimizes P_f for a given tolerated value of P_e.

Let us derive the threshold T assuming that the channel is AWGN, of variance σ^2. For other kinds of symmetric channels, such as the UMTS channel, an equivalent AWGN channel may be estimated. Code bits are assumed to be mapped into a BPSK symmetric signaling such that $r_n = \pm\sqrt{E_b}$. For such channel, when x_n is considered at the decoder output and y_n is received, (23) gets

$$\Lambda_n = \frac{2\sqrt{E_b}}{\sigma^2} y_n.$$

Thus, (21) and (22) become

$$y_n \underset{A_1^n}{\overset{A_{0|1}^n}{\lessgtr}} \frac{\sigma^2}{2\sqrt{E_b}} T \text{ and } y_n \underset{A_{0|1}^n}{\overset{A_0^n}{\lessgtr}} -\frac{\sigma^2}{2\sqrt{E_b}} T$$

Then P_e and P_f are expressed as

$$\begin{aligned} P_e &= \frac{1}{2}\left(1 - \text{erf}\left(\sqrt{\frac{E_b}{2\sigma^2}}\left(\frac{-T\sigma^2}{2E_b} + 1\right)\right)\right). \\ P_f &= \text{erf}\left(\sqrt{\frac{E_b}{2\sigma^2}}\left(\frac{T\sigma^2}{2E_b} + 1\right)\right) + \text{erf}\left(\sqrt{\frac{E_b}{2\sigma^2}}\left(\frac{T\sigma^2}{2E_b} - 1\right)\right). \end{aligned}$$

where $\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$. One can finally express the threshold T as

$$T(\sigma, P_e) = 2\sqrt{\frac{E_b}{\sigma^2}}\left(\left(\sqrt{2}\text{erf}^{-1}\left(1 - P_e\right)\right) - \sqrt{\frac{E_b}{\sigma^2}}\right).$$

Several choices may be considered for P_e. We constrain the probability of making a wrong decision to be a fraction of the probability $P_{\text{Hard}}(\sigma)$ that the hard decoder locally fails, expressed in the AWGN case by

$$\begin{aligned} P_{\text{Hard}}(\sigma) &= \int_0^{+\infty} P(y|X_n = 0)dy + \int_{-\infty}^{0} P(y|X_n = 1)dy. \\ &= 1 - \text{erf}\left(\sqrt{\frac{E_b}{2\sigma^2}}\right). \end{aligned} \tag{24}$$

Then, P_e can be expressed as

$$P_e(\sigma, n) = \alpha \cdot P_{\text{Hard}}(\sigma, n), \text{with } \alpha < 1. \tag{25}$$

The parameter α allows one to adjust the decoding complexity-efficiency tradeoff, as shown in Section V.

V. SIMULATIONS

Simulation are performed using the CABAC defined in the H.264/AVC standard. Binarized source symbols belong to the first 9 binary codewords of the zero-order Exp-Golomb scheme (EG0) [17]. A simplified context modeling with three contexts is considered.

Simulations using an AWGN channel and Pedestrian-B UMTS soft error patterns, see, *e.g.*, [30] are considered. For the AWGN channel, no channel coding is used and error correction relies only on the redundancy due to the EG0 binarization scheme (see Figure 1). For the Pedestrian-B channel, the CABAC is followed by a rate $1/2$ convolutionnal code with constraint length 9 and generators $(561, 753)_{\text{o}}$. A standard UMTS interleaver of length 640 is also considered. At the channel output, a SOVA decoder is implemented, providing log-likelihood ratios (LLR_n).

The Symbol Error Sate (SER) is evaluated for different values of the SNR. Hard decoding provides the bit value x_n using the sign of the channel output y_n in the AWGN case, and of the LLR_n in the UMTS case. Hard decoding fails if debinarization fails or if the EOS is not decoded from this bitstream (erasure). When an erasure occurs, the decoder does not output any solution and all symbols emitted by the source are considered as erroneous, and are included in the SER evaluation.

A. Performances of the MAP estimator

Figure 2 compares the results obtained using the proposed exact MAP decoder and the one based on the metric proposed by [13] on specific source sequences. These source sequences contain 100 binarized source symbols and every sequence is transmitted within packets of 640 bits. They were chosen in such a way that the number of *follow-on* procedures performed during encoding and the mean value of $follow$ is much higher than for totally random sequences. An M-algorithm with $M = 10$ is used for decoding. For a SER of $10^{-3},$ an improvement of 0.6 dB is achieved. The purpose, here is to show that the proposed MAP estimator derivation is, indeed more efficient than the previously proposed ones.

It is clearly seen on Figure 2 that the exact computation may lead to improved decoding in some situations. Note that the approximation used in the previous works is often quite accurate, and that, on many sequences, both computations provided almost the same performance. However, as illustrated, on some sequences the phenomenon shown on Figure 2 was observed. This clearly illustrates the usefulness of using an exact computation rather than an approximate one.

Figure 3 illustrates the results obtained using source sequences of 100 binarized source symbols and the M-algorithm with $M = 20$. The performance of a MAP decoder using (4) is compared to that obtained using a Maximum-Likelihood (ML) decoder. For a SER of 10^{-3}, the improvement (in dB) obtained when using (4), compared to the ML metric is about 0.7 dB for both channels. The gain achieved by the soft decoding when compared to the hard one is up to 4 dB for the AWGN channel and 2 dB for the UMTS channel.

B. Adjustment of the complexity-efficiency tradeoff

MAP decoders based on both SA and MA are considered, and the following abbreviations are adopted. BSA and BMA(M) stand for the basic SA and MA, M being the number of paths kept at each MA iteration. GSA(ℓ) is the generalized SA, extending ℓ paths at each iteration. Finally FGSA(ℓ, α) and FMA(M, α) denote the Fast SA and the Fast MA embedding the test presented in Section IV-D.

Figure 5 illustrates the error resilience, in terms of SER, and complexity performance, in terms of average number of visited branches during erasure-free decoding, of four versions of the decoder using the SA. Figure 6 presents the same performance for four versions of the decoder using the MA.

When compared to a standard decoder carrying out hard decisions on noisy bits, the sequential decoders present an important gain (up to 3 dB for the SA based decoders and 4 dB for the MA) in error correction. The performance is improved as the number of simultaneously explored paths (ℓ and M) increases, and as α decreases.

On the other hand, one may notice that the more efficient is the decoding in recovering errors, the higher is the complexity. Compared to the BSA, the FGSA($3, 10^{-4}$) reaches a gain of 1.8 dB, at SER $= 10^{-3}$ for a doubled complexity at 12 dB. For the same SNR, FMA($20, 10^{-4}$) presents a 20 times lower complexity when compared to MA(10), with a gain of 2.5 dB at SER $= 10^{-3}$. This shows that the effects of α and ℓ or M on the tradeoff between the complexity and the robustness may be combined in order to design JSCD schemes according to application needs.

Figure 4 depicts the packet erasure rate related to decoding failures for the same versions of the sequential algorithms. For the FGSA and FMA decoders, packet losses are mainly caused by the dropping strategy introduced in Section IV-D. Indeed, for low values of the SNR, the number of stored paths often reaches zero before any solution is obtained. Nevertheless, one can note, for example, that for a SNR of 10 dB, FGSA($3, 10^{-4}$) presents a packet loss

rate 200 times lower than the BSA. This avoids too frequent use of ARQ. For instance, supposing that the channel rate is equal to 64 kb/s, the effective rate available to the source, at SNR $= 9$ dB and taking the ARQ into account, is 57.6 kb/s using a BSA, 63.7 kb/s using a FGSA($3, 10^{-4}$), 62.08 kb/s using a MA(10), and 63.93 kb/s using a FMA($20, 10^{-4}$).

VI. Conclusions

The main drawback of the high compression rate CABAC is its vulnerability to transmission errors. In this paper, we have presented a soft decoding technique based on a MAP estimator, exploiting binarization scheme, information given by context modeling, and the semantic of the code string. This estimator is associated to sequential decoding algorithms to carry out a reliable soft CABAC decoder. An objective test allowing to adjust the decoding complexity according to the desired performances is elaborated, and shown to provide better performance in terms of symbol error rates for a given complexity.

In all variants of the decoder, the small redundancy present after the binarization step is exploited and no extra redundancy is added. The proposed sequential decoders can handle adaptive probabilities and context modeling, and can be directly embedded into a standard implementation of CABAC without impairing its compression efficiency. Current work is dedicated to embedding the soft MAP decoder and the objective test within the full H.264 decoder.

References

[1] S. Ben Jamaa, M. Kieffer, and P. Duhamel, "Exact map decoding of cabac encoded data," *submitted to Proceedings of ICASSP*, 2006.

[2] S. Ben-Jamaa, M. Kieffer, and P. Duhamel, "Controlled complexity map decoding of cabac encoded data," 2006.

[3] M. Park and D. J. Miller, "Joint source-channel decoding for variable length encoded data by exact and approximate MAP sequence estimation," *IEEE Trans. on Comm.*, vol. 48(1), pp. 1–6, 2000.

[4] K. Sayood, H. H. Otu, and N. Demir, "Joint source-channel coding for variable length codes," *IEEE Trans. on Comm.*, vol. 48(5), pp. 787–794, 2000.

[5] M. Bystrom, S. Kaiser, and A. Kopansky, "Soft source decoding with applications," *IEEE Trans. on Circuits Syst. Video Technol.*, vol. 11(10), pp. 1108–1120, 2001.

[6] J. Wen and J. Villasenor, "Soft-input soft-output decoding of variable length codes," *IEEE Trans. on Comm.*, vol. 50(5), pp. 689–692, 2002.

[7] P. G. Howard and J. S. Vitter, "Practical implementations of arithmetic coding," *Image and Text Compression*, vol. 13(7), pp. 85–112, 1992.

[8] C. Boyd, J. Cleary, I. Irvine, I. Rinsma-Melchert, and I. Witten, "Integrating error detection into arithmetic coding," *IEEE Trans. on Comm.*, vol. 45(1), pp. 1–3, 1997.

[9] J. Sayir, *On Coding by Probability Transformation*, Ph.D. Thesis Nr. 13099, EE Department, ETH Zurich, Switzerland, 1999.

[10] J. Chou and K. Ramchandran, "Arithmetic coding-based continuous error detection for efficient ARQ-based image transmission," *IEEE Trans. on Comm.*, vol. 18(6), pp. 861–867, 2000.

[11] B. D. Pettijohn, W. Hoffman, and K. Sayood, "Joint source/channel coding using arithmetic codes," *IEEE Trans. on Comm.*, vol. 49(5), pp. 826–836, 2001.

[12] T. Guionnet and C. Guillemot, "Soft decoding and synchronization of arithmetic codes: Application to image transmission over noisy channels," *IEEE Trans. on Image Processing*, vol. 12(12), pp. 1599–1609, 2003.

[13] M. Grangetto, P. Cosman, and G. Olmo, "Joint source/channel coding and MAP decoding of arithmetic codes," *IEEE Trans. on Comm.*, vol. 53(6), pp. 1007–1016, 2005.

[14] C. Demiroglu, W. Hoffman, and K. Sayood, "Joint source channel coding using arithmetic codes and trellis coded modulation," *Proc. of DCC, Snowbird, Utah, USA.*, pp. 302–311, 2001.

[15] B. Dongsheng, W. Hoffman, and K. Sayood, "State machine interpretation of arithmetic codes for joint source and channel coding," *Proc. of DCC, Snowbird, Utah, USA.*, pp. 143–152, 2006.

[16] L. R. Bahl, J Coke, F. Jelinek, and R. Ravid, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on Inform. Theory*, vol. 20(2), pp. 284–287, 1974.

[17] D. Marpe, H. Schwarz, and T Weigand, "Context based adaptative binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13(7), pp. 620–636, 2003.

[18] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE *Trans. on Circuits and Systems for Video Technology*, vol. 13(7), pp. 560–576, 2003.

[19] S. Saponara, C. Blanch, K. Denolf, and J. Bormans, "The JVT advanced video coding standard complexity and performances analysis on a tool-by-tool basis," *IEEE Packet Video*, 2003.

[20] J. B. Anderson and S. Mohan, *Source and channel coding: an algorithmic approach*, Kluwer Academic Publishers, Norwell, MA, 1991.

[21] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. on Information Theory*, vol. 6(3), pp. 194–203, 1975.

[22] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30(6), pp. 520–540, 1987.

[23] J. Wen and J. D. Villasenor, "Utilizing soft information in decoding of variable length codes," *Proc. of DCC, Snowbird, Utah, USA.*, pp. 131–139, 1999.

[24] L. Guivarch, J. C. Carlach, and Siohan P., "Joint source-channel soft decoding of huffman codes with turbo-codes," *Proc. of DCC, Snowbird, Utah, USA.*, pp. 83–92, 2000.

[25] M. Jeanne, *Etude de systmes robuste de dcodage conjoint source-canal pour la transmission sans fil de vido*, PhD Thesis Nr. D 03-18, France Telecom R&D, Rennes, 2003.

[26] K. Zigangirov, "Some sequential decoding procedures," *Probl. Peredach. Inform.*, vol. 2(4), pp. 13–15, 1966.

[27] F. Jelinek, "Fast sequential decoding algorithm using a stack," *IBM J. Res. Develop.*, vol. 13, pp. 675–685, 1969.

[28] D. Haccoun and M. J. Ferguson, "Generalized stack algorithms for decoding convolutional codes," *IEEE Trans. on Information Theory*, vol. 21(6), pp. 638–651, 1975.

[29] P. R. Chevillat and D. J. Costello, "A multiple stack algorithm for erasurefree decoding of convolutional codes," *IEEE Trans. on Comm.*, vol. 25(12), pp. 1460–1470, 1977.

[30] M. Jeanne, I. Siaud, O. Seller, and P. Siohan, "Application of a joint source-channel decoding technique to UMTS channel codes and OFDM modulation," *Proc. of ICT, Fortaleza, Brazil*, pp. 912–923, 2004.

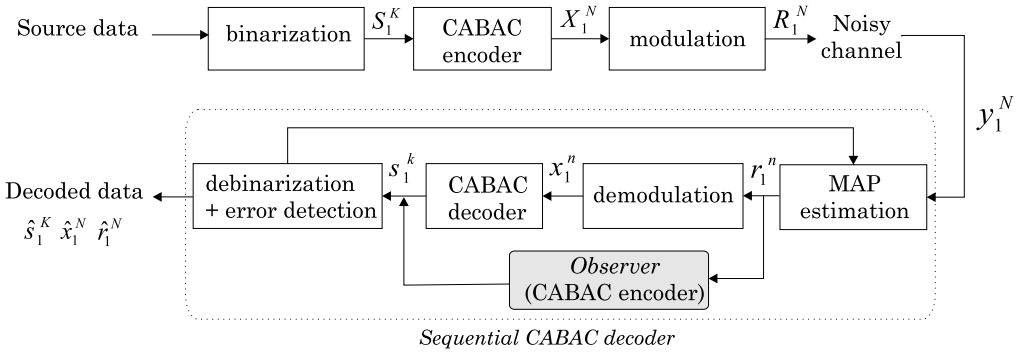

Fig. 1. Transmission block diagram

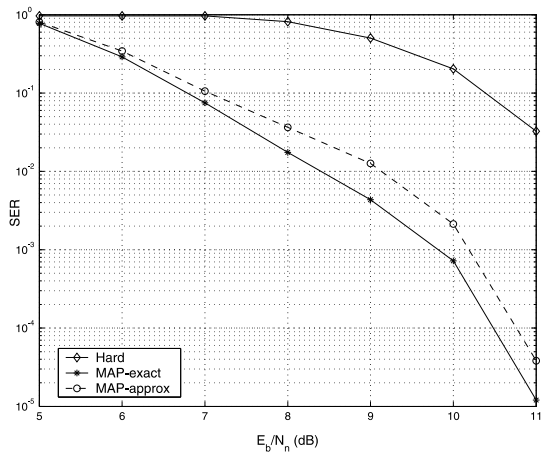

Fig. 2. Performance of the proposed MAP vs. the approximated MAP proposed in citeGrangetto05

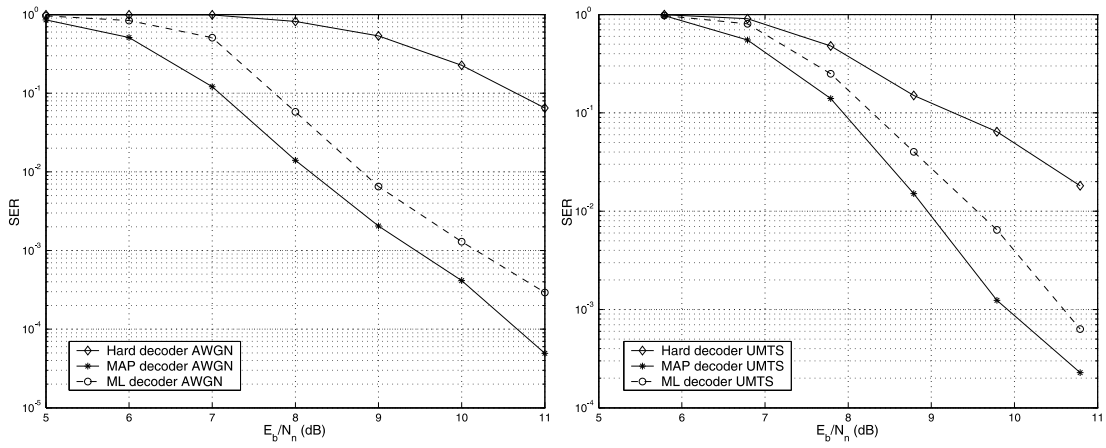

Fig. 3. MAP performance vs. hard and ML decoding for AWGN channel (without channel coding), and UMTS channel (with convolutional coding).

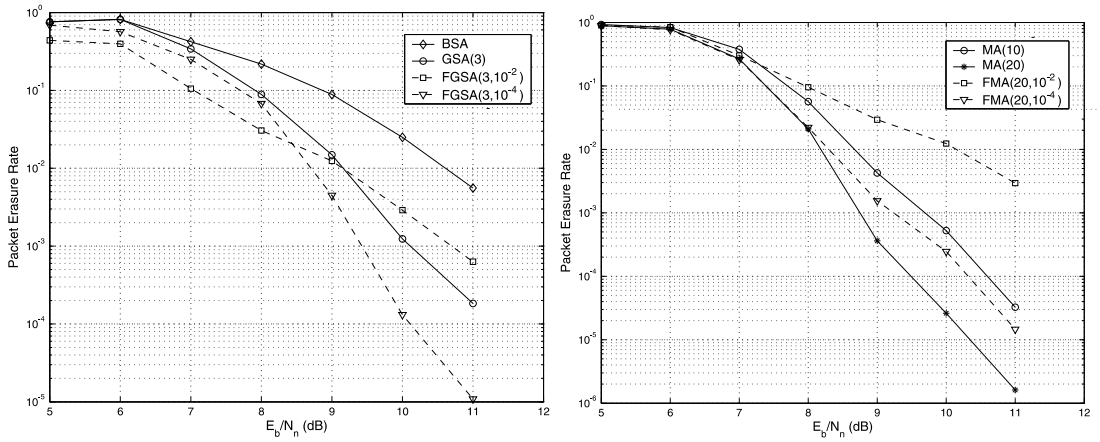

Fig. 4. Packet erasure rate using SA and MA based decoders

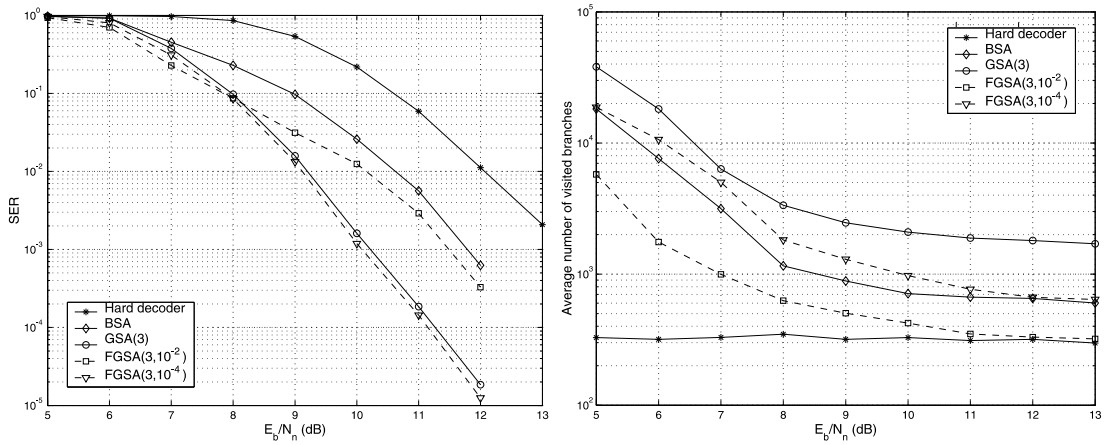

Fig. 5. Performance in terms of error resilience and complexity of the decoder using the SA

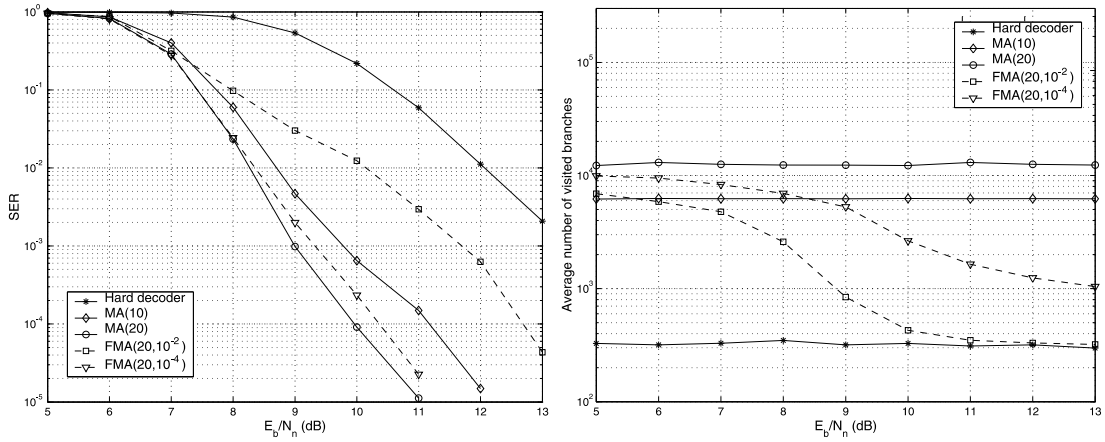

Fig. 6. Performance in terms of error resilience and complexity of the decoder using the MA